The Monte Carlo Method

IB Math HL Period 5, Dr. Silverman

Jackson Chen

16 November 2015

1 Introduction

1.1 Personal Engagement

I have always learned to solve numeric problems, such as calculating the area of a shape in Euclidean geometry, with a deterministic formulaic method. This may consist of using numeric relationships or calculus. However, I believed that the conventional way to solve these problems were limited in that it was difficult and time-consuming to calculate the areas of irregular shapes. Furthermore, I am a technology enthusiast, and am fascinated by how the rapidly growing computational power of computers is influencing many topics within math and science. However, I saw geometry as an area that lagged behind other math topics until a professor introduced me to the Monte Carlo method. I had never thought of applying probability to numeric problems with seemingly definite solutions until then. The Monte Carlo method will be described in more detail in Section 1.2. This method is also surprisingly straightforward compared to some of the other methods used to solve numeric problems, and may have even an advantage when finding areas of irregular shapes.

1.2 Explanation

The Monte Carlo method consists of using random sampling in order to solve numerical problems. This may seem like an imprecise method of solving, for example, the area of a circle. However, there are two crucial criteria to consider when using the Monte Carlo method:

- The greater the number of random inputs that are chosen, the greater the accuracy
- The more uniform the distribution of random samples, the more accurate the final answer

In order to address the second point made in the list above, one may assume that truly random numbers are needed to ensure a fair distribution. However, this is not the case as pseudorandom number generation, which is prevalent in many number generator algorithms, should be satisfactory for the usage of the Monte Carlo method.

2 History

Before the Monte Carlo method, statistical sampling was used to estimate uncertainties in simulations. The Monte Carlo method reversed this approach, using probabilistic analogs to solve deterministic problems.[1] Probabilistic analogs use a heuristic technique to find or generate sufficient solutions to optimization problems. These techniques do not guarantee that the best solution will be found, but they use less computation time and power than optimization algorithms or iterative techniques.

The earliest variant of the Monte Carlo method can be seen in the Buffon's needle experiment in the 18th century where π was estimated by dropping needles on a floor made of equidistant and parallel strips.[2]

However, the current version of the Monte Carlo method was created in the late 1940's by Stanislaw Ulam while working on a nuclear weapons project. He was trying to investigate radiation shielding and the distance neutrons would likely travel through various materials. Despite his team obtaining all of the necessary data, the physicists were unable to solve the problem using a conventional deterministic method. Ulam thought of using random experiments. He generated the pseudorandom numbers using the middle-square method, the quickest at his disposal. The middle-square method consists of creating a seed number, squaring it, and then using the middle n digits as the pseudo-randomly generated number. Monte Carlo was the code name of the work, due to the level of secrecy required, and the name stuck.[3]

The Monte Carlo methods were widely used in the Manhattan Project, and used in the development of the hydrogen bomb. The Rand Corporation and the US Air Force widely funded work on Monte Carlo methods during this time, broadening its applications to various other fields.

3 Calculation

A sample calculation for the Monte Carlo Method will be performed to calculate the area of a circle. This will be done to demonstrate the properties of the method.

[1] "The Beginning of the Monte Carlo Method" by N. Metropolis, 1987
[2] Ibid.
[3] "Stan Ulam, John von Neumann and the Monte Carlo Method" by Roger Eckhardt, 1987

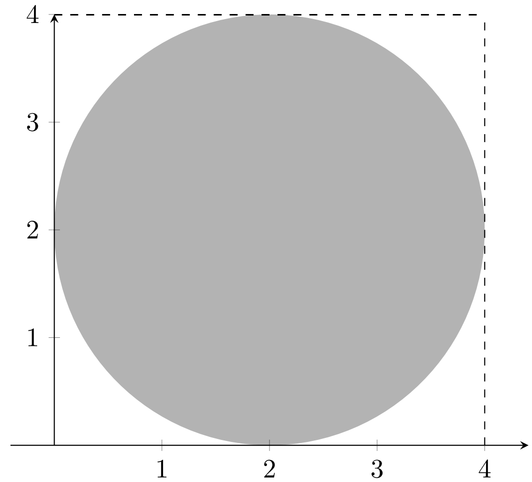

Figure 1: A circle with center (2, 2) and radius 2 used in the sample calculation

The circle is centered at (2, 2) and has radius 2. Thus the equation for the circle is given with

$$(x-2)^2 + (y-2)^2 = 4$$

The domain of possible inputs consists of the square bounded by the x and y axes and the lines $x = 4$ and $y = 4$ as represented by the dashed lines in Figure 1.

Using the standard area of a circle equation, the exact value for the area can be found:

$$A = \pi r^2 = 4\pi \approx 12.566$$

Now to calculate the area using the Monte Carlo method, I created a computer program that will be used to generate several thousand points within the domain. For every "number of test points" case, 1000 trials were done and the answer averaged. The purpose of doing that many trials is to normalize any number generation anomalies. The code for the program will be shown in the appendix.

The program uses the `random` Python library, which implements psuedo-random number generation. This library generates random floats uniformly in the range $[0, 1)$. Since Python uses Mersenne Twister as the core generator, it produces 53-bit precision floats with a period of $2^{19937} - 1$.[4] Despite the fact that the generator is not perfectly random, the `random` library satisfies the second criterion mentioned in Section 1.2.

[4] https://docs.python.org/2/library/random.html

Number of test points	Points in circle	Percent of Domain	Predicted Area	Percent Error
10	7	70.0%	11.200	10.9%
20	15	75.0%	12.000	4.50%
50	39	78.0%	12.480	0.684%
100	78	78.0%	12.480	0.684%
500	393	78.6%	12.576	0.0796%
1000	785	78.5%	12.560	0.0477%
2000	1570	78.5%	12.560	0.0477%
5000	3927	78.5%	12.560	0.0477%
10000	7850	78.5%	12.560	0.0477%
50000	39272	78.5%	12.560	0.0477%
100000	78539	78.5%	12.560	0.0477%

Table 1: Predicted area and its accuracy for a varying number of test points.

It took a computer approximately 101.5 seconds to generate 100000 points. Thus, the maximum test points generated was capped at 100000 for the sake of time and because additional points were yielding negligible changes in predicted area accuracy.

The predicted area is calculated by taking the percent of the domain (which is the percent of the total random samples that are in the circle) and multiplying it by the area of the domain, in this case, a square with area 16.

The data in 2 reflects the first criterion in Section 1.2. The greater the number of test points, the smaller the percent error and thus the more accurate the predicted area is. Once enough points were used, the percent error became less than half of a percent.

In the previous situation, it seemed disadvantageous to use the Monte Carlo method in order to calculate the area of the sphere. It requires more computing time and power than using the deterministic numerical calculation. This may be true for simple cases such as that. However, in the cases of irregularly shaped polygons or integration with hundreds of degrees of freedom (or dimensions), the curse of dimensionality [5] dictates an exponential increase in complexity as the number of dimensions rises when using deterministic numerical integration. Using the Monte Carlo method instead would prevent the exponential increase in computational time. A more complicated example involving integration will be provided to demonstrate the ease of the Monte Carlo method compared to conventional calculations.

The area enclosed by two quadratic functions as shown in Figure 2 will be calculated using the conventional method (with integrals) and with the Monte Carlo method. This example will show how the Monte Carlo method is much simpler than subtracting integrals. The dashed box at $x = 6$ and $y = 6$ represents the domain of the Monte Carlo simulation.

[5] Richard Ernest Bellman; Rand Corporation (1957). Dynamic programming. Princeton University Press.

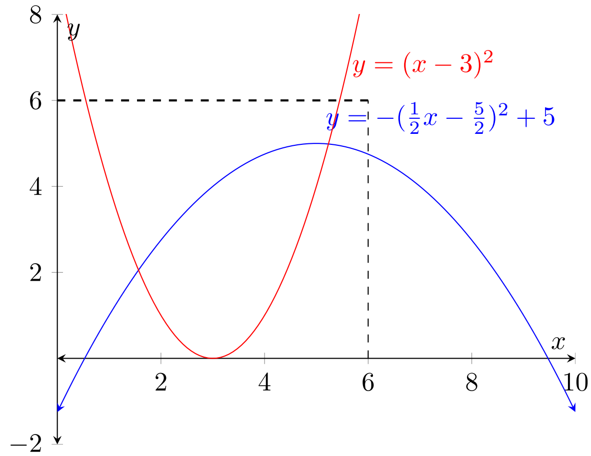

Figure 2: A graph with two intersecting quadratic functions. The enclosed area in the middle is in question.

Solving with integration

First the limits of integration (the two intersection points) need to be determined.

$$(x-3)^2 = -\left(\frac{1}{2}x - \frac{5}{2}\right)^2 + 5$$

$$x = 1.567, 5.233$$

Now the formula for the area can be set up.

$$A = \int_{1.567}^{5.233} \left(-\left(\frac{1}{2}x - \frac{5}{2}\right)^2 + 5\right) dx - \int_{1.567}^{5.233} (x-3)^2 dx$$

$$A = \int_{1.567}^{5.233} [(-\left(\frac{1}{2}x - \frac{5}{2}\right)^2 + 5) - (x-3)^2] dx$$

$$\boxed{A = 10.265}$$

Solving with the Monte Carlo Method

It took a computer approximately 104.6 seconds to generate 100000 points. As shown in Table 2, the percent error diminishes as more and more data points are generated. At 100000 data points, the percent error is negligible. However, there are slight variations in the data, for example the area predicted by 10000 data points was less accurate than that of 5000 data points; this is due to the varying nature of pseudo-random number generation.

Number of test points	Points in Area	Percent of Domain	Predicted Area	Percent Error
10	2	20.0%	7.200	29.900%
20	5	25.0%	9.000	12.323%
50	14	28.0%	10.080	1.802%
100	28	28.0%	10.080	1.802%
500	141	28.2%	10.152	1.101%
1000	285	28.5%	10.260	0.0487%
2000	568	28.4%	10.224	0.3994%
5000	1426	28.52%	10.267	0.0214%
10000	2849	28.49%	10.256	0.0838%
50000	14256	28.51%	10.264	0.0066%
100000	28514	28.51%	10.265	0.0004%

Table 2: Predicted area and its accuracy for a varying number of test points.

It can be said that in this example, where the area between two curves is calculated, it is much simpler and less time-consuming to use the Monte Carlo method rather than using integration. Setting up the integration (determining the limits of integration) and solving it requires a hefty amount of calculation, while generating 100000 points with a computer program using the Monte Carlo method resulted in the same answer.

4 Applications

The Monte Carlo method has a wide array of applications in science, engineering, and computing. For example, it is used in ensemble models to help with weather forecasting. In this case, multiple predictions are made with slightly different initial conditions that may result from past observations. The multifarious simulations are used to account for errors that would be introduced through the use of imperfect conditions and imperfections in the model formulation. In addition, this type of simulation is also sometimes used during wartime in order to calculate the position of enemy aircraft in the skies.

Computer graphics use Monte Carlo Ray Tracing to render three dimensional scenes by randomly tracing samples of paths of light. This can be used in applications like first person video games, where Monte Carlo simulations help determine if the enemies or objects behind walls can be seen from the perspective of the player through path tracing. Simulations with a massive amount of samples on pixels allow the sample average to converge to the solution of the rendering solution, thus making Monto Carlo simulations one of the most accurate three dimensional graphics rendering methods.

In addition to rendering games, Monte Carlo Tree Search (MCTS) has allowed for bots for games to search for the best move in any given situation. MCTS has been successfully used in Go, Tantrix, and Battleship.

5 Conclusion

The Monte Carlo method provides an alternative to the conventional deterministic numerical computations used to solve a vast array of math problems, including area calculation and integration. In situations where the complexity of the problem exponentially grows when using deterministic methods, such as using numerical integration for functions in many dimensions, the Monte Carlo method typically becomes advantageous.

The Monte Carlo method has a plethora of applications that revolutionized various fields in science, as well as human wartime tactics. It provides the most accurate way of rending three dimensional graphics, revolutionized path finding algorithms for computer bots, and allows for the prediction of weather that accounts for the uncertainty unsolved for in conventional methods.

Through investigating Monte Carlo simulations, I have grown as a mathematician, in terms of broadening my ability to solve deterministic problems in an unconventional manner. With my new knowledge, I can fully appreciate the widespread applications that math entails in my daily life. The multidisciplinary approach taken with this method has driven me to continue researching this topic, as I have experience in many fields of that have been influenced by this method.

A Appendix: Python Code for the Monte Carlo Program

```
import random
import time

points = 100000
cumSum = 0
n = points

start = time.time()
for j in range(1000):
    pointsOut = 0
    pointsIn = 0
    x = [random.random()*6 for i in xrange(n)]
    y = [random.random()*6 for i in xrange(n)]
    for i in range(n):
        if (y[i] < (-(0.5*x[i]-2.5) ** 2 + 5) and y[i] > (x[i]-3) ** 2):
            pointsIn += 1
    cumSum += pointsIn

f = open('result.txt', 'w')
f.write(str(cumSum/1000))
f.close()
end = time.time()
print end - start
```